Quantization -> Higher memory representation to lower memory representation

Normally all the weights and parameters are stored in FP32 format -> 32 bit
This is called **Full precision / single precision**

We can convert this 32 bit, maybe into an 8-bit value. This reduces the memory usage of the model and parameters.

Quantization can be done not just in the case of llms, but any deep learning model

Post Training quantization and Quantization aware training

-> inference
-> edge devices deployment

FP 32 bit -> FP 16 bit is called **Half-precision**

Symmetric unit8 quantization

-> All the data is evenly distributed within itself

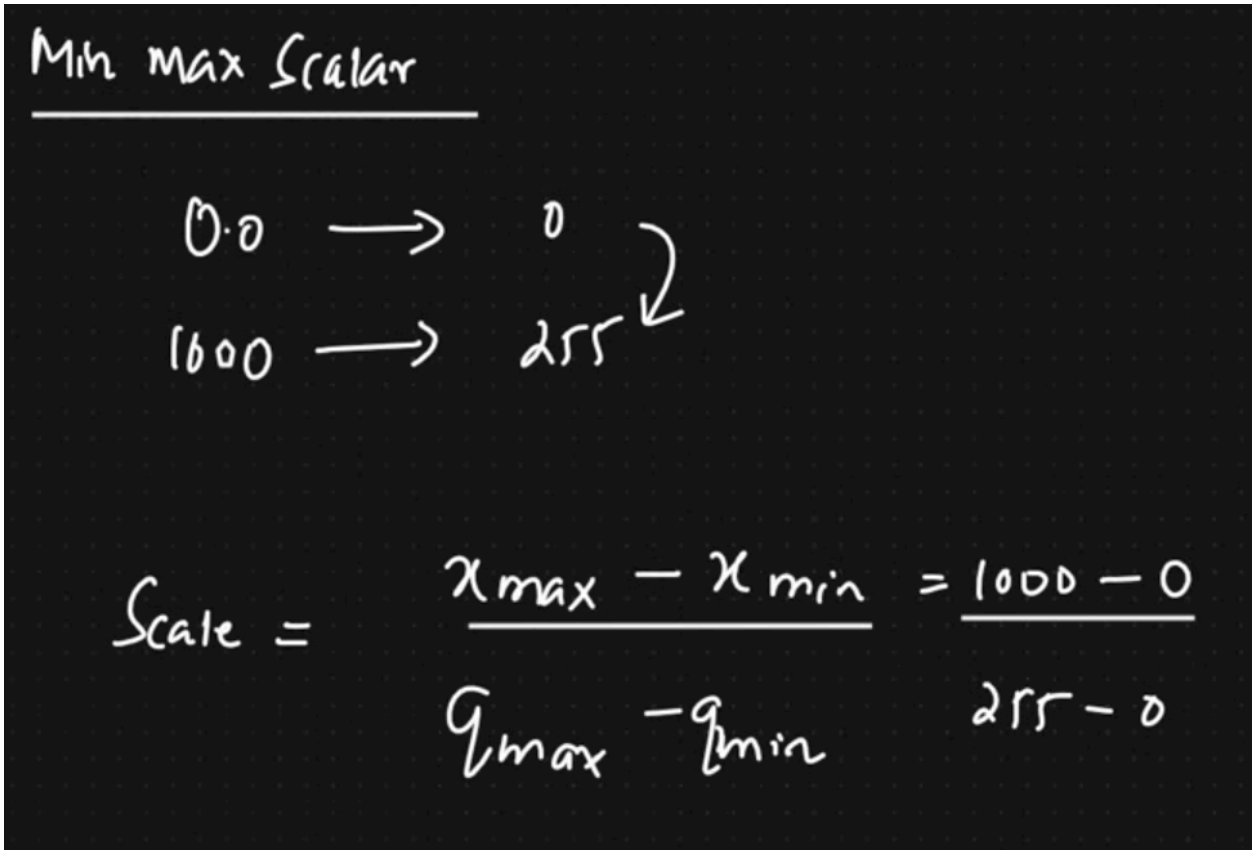
Assume a value lies in the range 0 to 1000

We want to store this value in a uint8

So, we have to store all these ranges into an uint8. So it has to be in the range 0-255.

How to scale this?

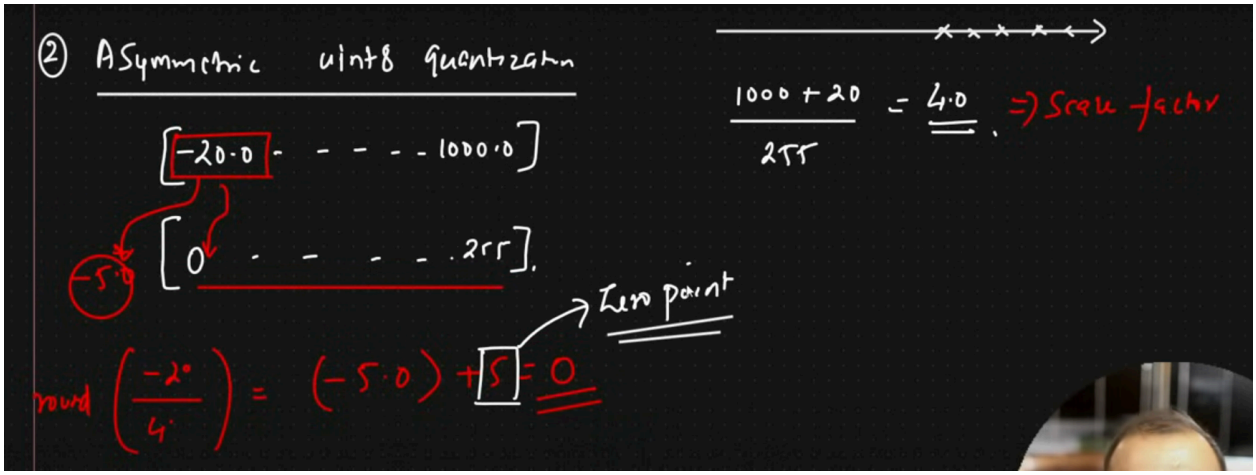
Min max scaler:



This gives us a scaling factor 3.92. So, we divide the values of the original range with the scaling factor and round them to bring the values to the required range

Asymmetric unit8 quantization:

It is not symmetric. So, things are little different in here.



This zero point is 0 in the case of a symmetric distribution. In the case of asymmetric distribution, we add the same value we get by dividing the scaling factor.

This squeezing process of higher range to a lower range is called “CALIBRATION”.

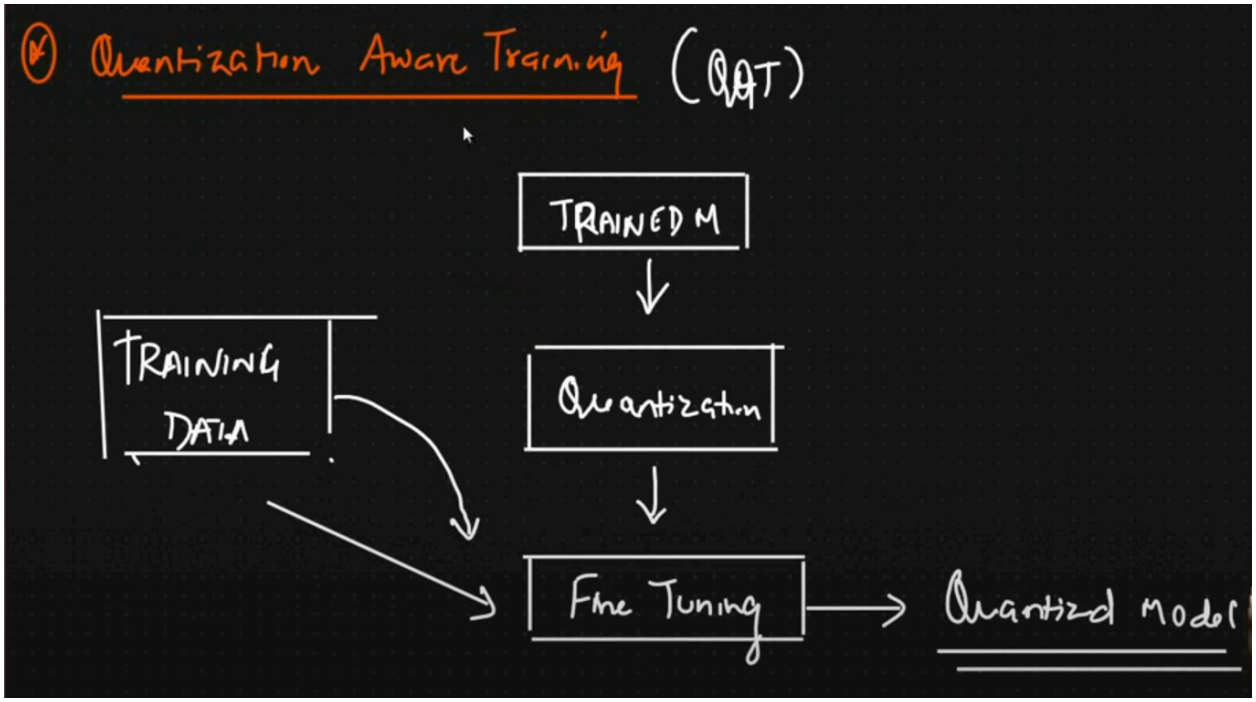
Modes of Quantization:

Post-Training quantization:

- We already have a pre-trained model
- We then calibrate the weights data in the pre-trained model and we get the quantized model.
- We can use this quantized model in our usecases

Quantization Aware Training(QAT):

With post-training quantization, we understand that we lose some accuracy.

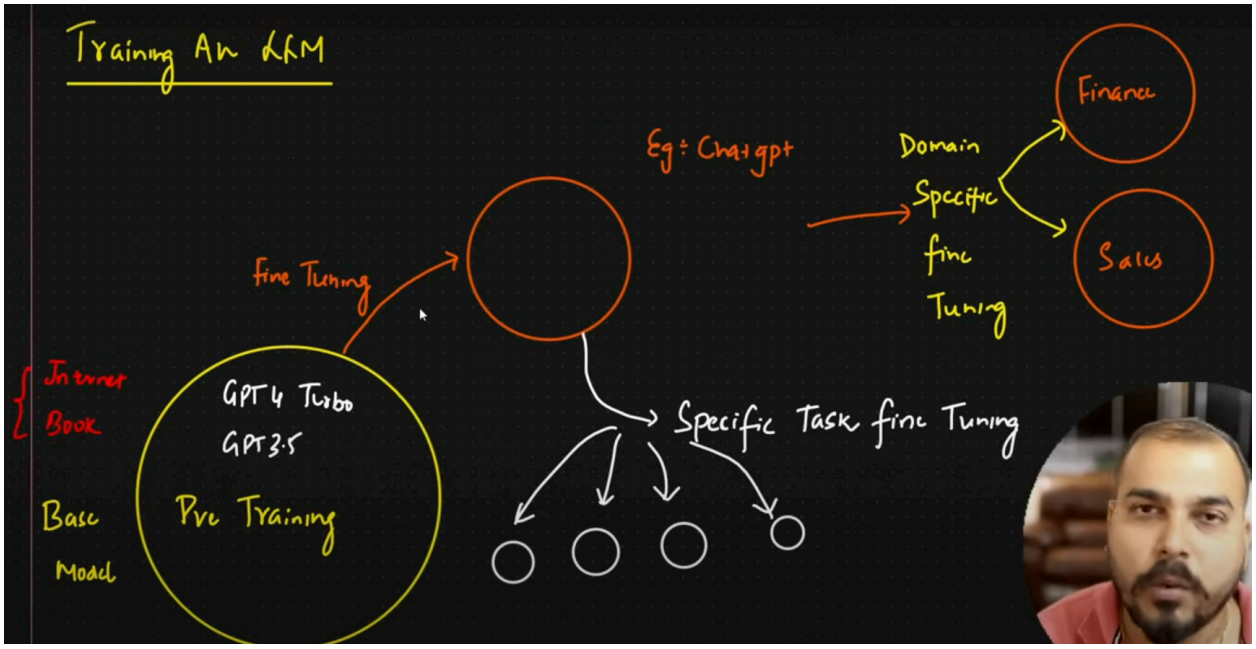


With any finetuning, we typically use QAT, so that we do not lose a lot of accuracy.

LoRA and QLoRA in-depth intuition:

When we have a pre trained LLM model like GPT-4 -> base model

This is trained on a huge amount of data.



Finetuning -> one way -> full parameter fine tuning

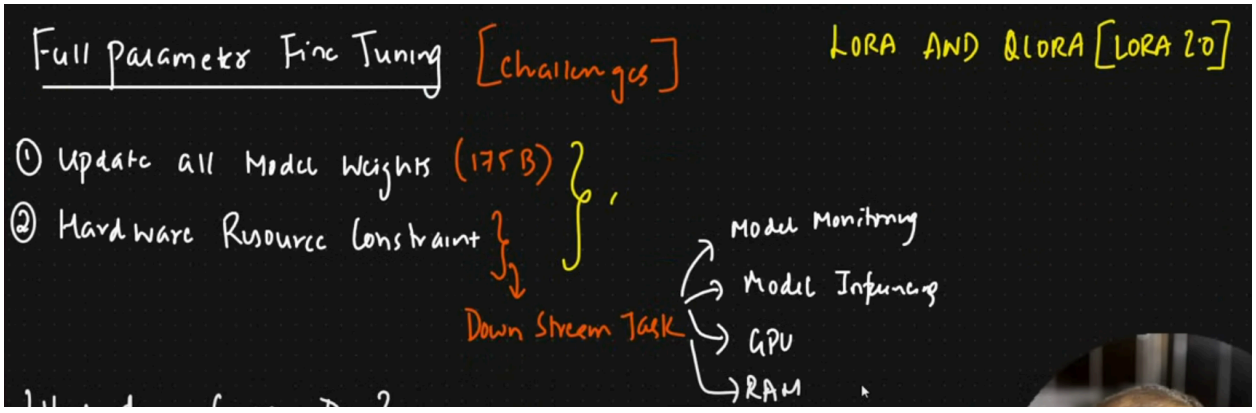
We can also do domain-specific finetuning or specific task fine-tuning

Each task can be a QandA task/ instruction-response task etc..

Full parameter Fine Tuning:

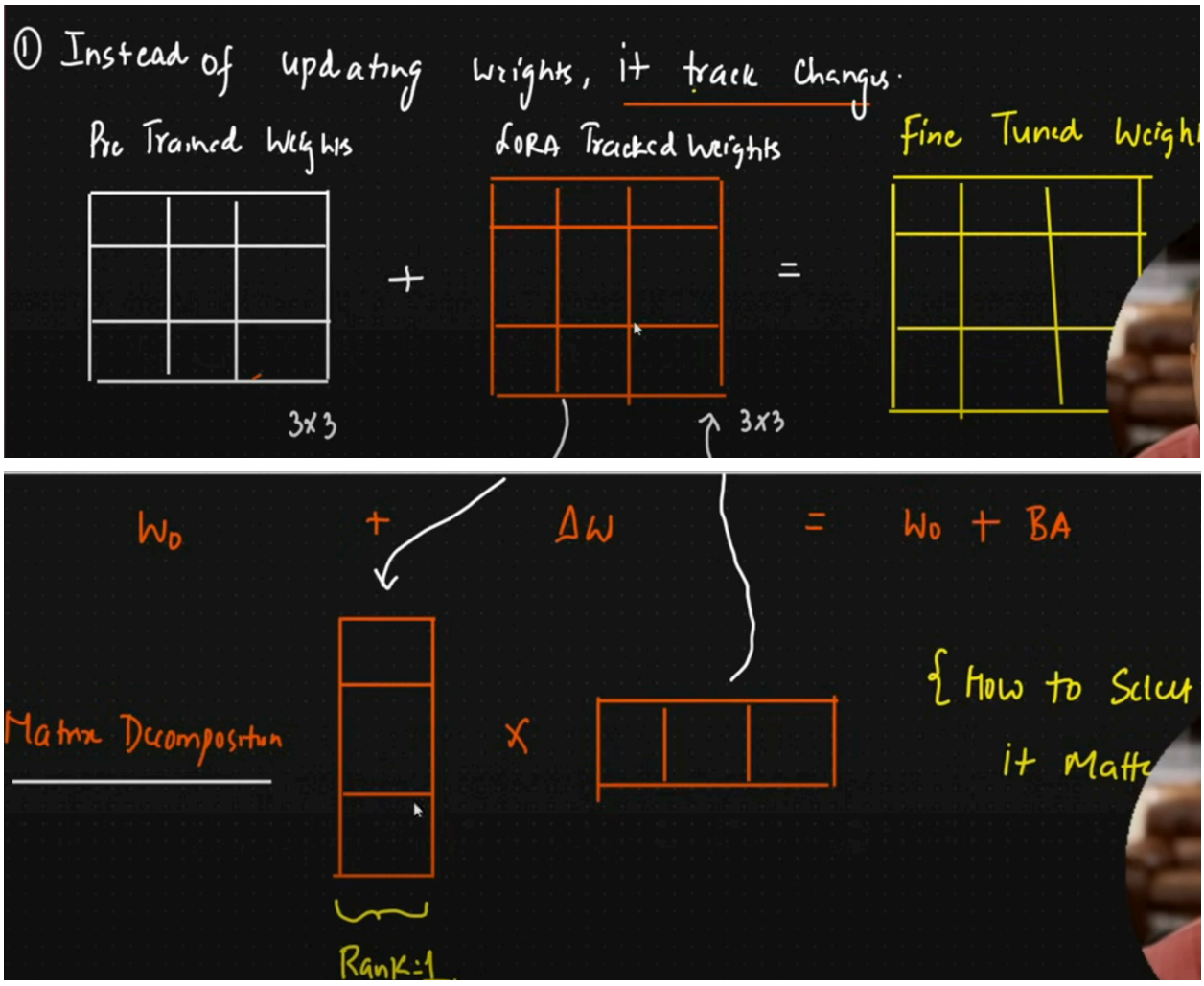
1) We need to update all the model weights
2) Hardware resource constraint

Challenges : Model monitoring, model inference, GPU, RAM etc.…



LoRA:

1) Instead of updating all the weights, it just tracks changes.

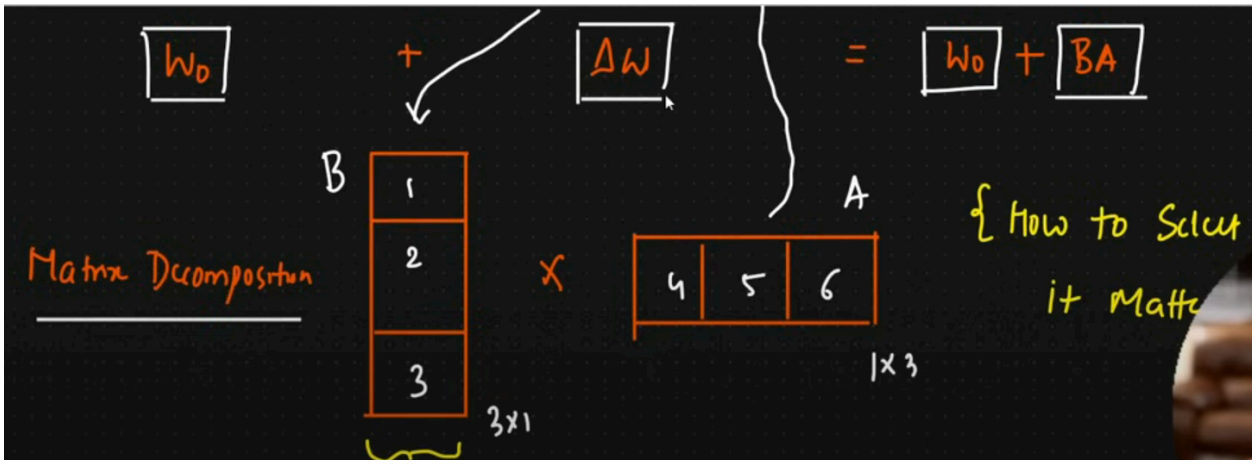


First we have pretrained weights from base model, and LoRA then tracks the changes in weights (red matrix), both are combined to get the finetuned weights

But, how does LoRA make it efficient?

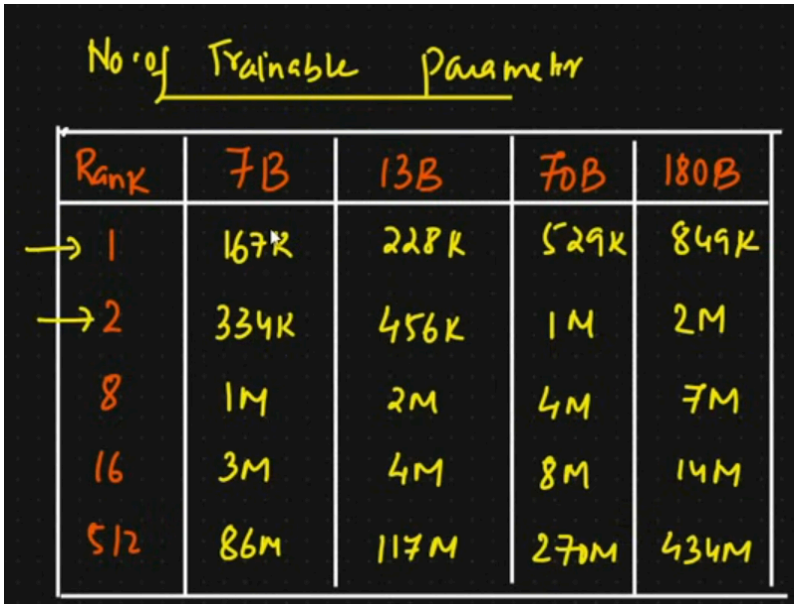
It performs **matrix decomposition**, so that the 9 values of tracking are converted into two smaller matrices 3x1 and 1x3. So, we are storing 6 values. When we multiply these both we get 3x3.

So, any big matrix is stored in the form of two smaller matrices and then these are used to get the values of tracking to get the finetuning model.



As the rank of these decomposed matrices increase, more values are stored, so number of parameters increase. But, still it is a lot better than original.

Method	Hyperparameters	# Trainable Param
Fine-Tune	-	175B
PrefixEmbed	$l_p = 32, l_i = 8$	0.4 M
	$l_p = 64, l_i = 8$	0.9 M
	$l_p = 128, l_i = 8$	1.7 M
	$l_p = 256, l_i = 8$	3.2 M
	$l_p = 512, l_i = 8$	6.4 M
PrefixLayer	$l_p = 2, l_i = 2$	5.1 M
	$l_p = 8, l_i = 0$	10.1 M
	$l_p = 8, l_i = 8$	20.2 M
	$l_p = 32, l_i = 4$	44.1 M
	$l_p = 64, l_i = 0$	76.1 M
Adapter^H	$r = 1$	7.1 M
	$r = 4$	21.2 M
	$r = 8$	40.1 M
	$r = 16$	77.9 M
	$r = 64$	304.4 M
LoRA	$r_v = 2$	4.7 M
	$r_q = r_v = 1$	4.7 M
	$r_q = r_v = 2$	9.4 M
	$r_q = r_k = r_v = r_o = 1$	9.4 M



QLoRA (Quantized-LoRA):

We just do the above LoRA and also quantize the values we store.

Example:

```
model_id = "google/gemma-2b"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

We are doing the quantization using BitsAndByterConfig. We are converting the entire thing into 4-bit. The quantization technique we are going to use is nf4.

```
lora_config = LoraConfig(
    r = 8,
    target_modules = ["q_proj", "o_proj", "k_proj", "v_proj",
                      "gate_proj", "up_proj", "down_proj"],
    task_type = "CAUSAL_LM",
)
```

Rank value -> 8

Fine Tuning Llama2 on a custom dataset:

PeFT -> parameter efficient transfer learning

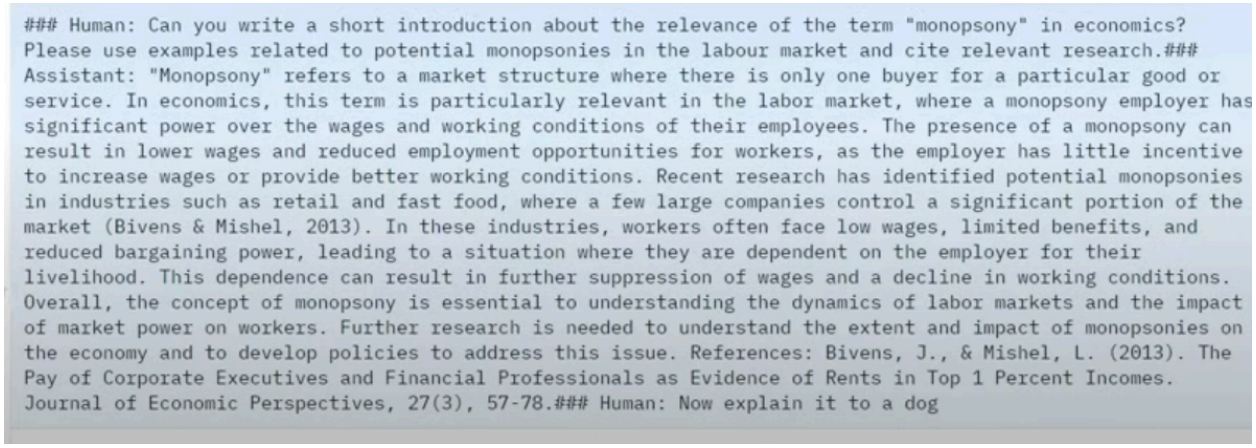
PeFT has LoRA and other stuff. It freezes most of the weights in the model and only certain values change.

Prompt template for the specific model that we are using.
Llama2 prompt template:

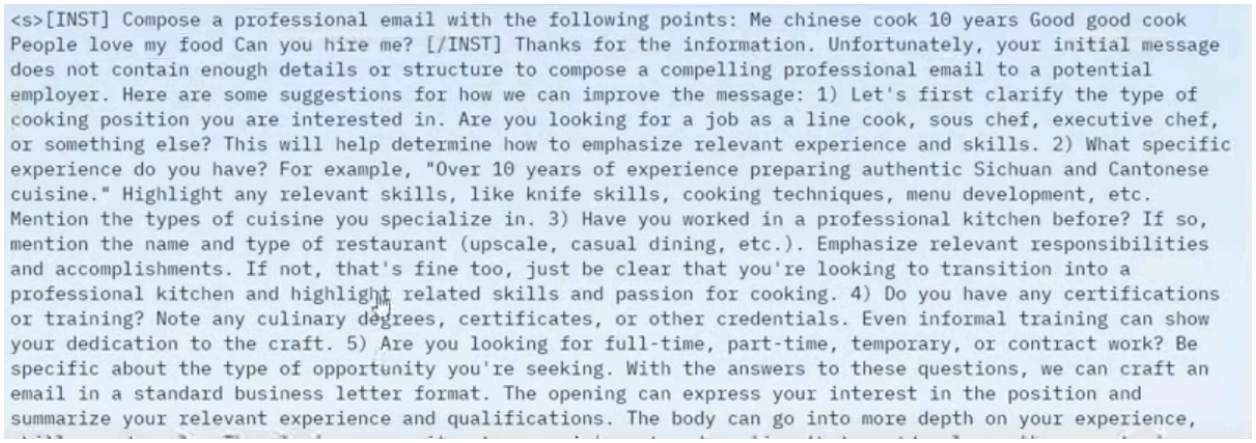
```
<s>[INST] <<SYS>>
System prompt
<</SYS>>

User prompt [/INST] Model answer </s>
```

Converting the dataset into the required prompt format for our model:



Original format in the dataset: Human and Assistant model
After reformatting 1k rows of the above dataset into format that our model needs.



Finetuning:

Llama2 has 7B parameters and hence, it is really difficult to do full finetuning as this would use very high hardware resources.

Task:

1. Load a llama-2-7b-chat-hf model (chat model)
2. Train it on the mlabonne/guanaco-llama2-1k (1,000 samples), which will produce our fine-tuned model Llama-2-7b-chat-finetune

QLoRA will use a rank of 64 with a scaling parameter of 16. We'll load the Llama 2 model directly in 4-bit precision using the NF4 type and train it for one epoch

Consider that this rank 64 and scaling parameter -alpha 16 are hypertuning parameters.